(19)日本国特許庁（ＪＰ）　(12) **公開特許公報（A）**　(11)特許出願公開番号

特開平8－263262

(43)公開日　平成８年(1996)10月11日

(51)Int.Cl.[6]	識別記号	庁内整理番号	ＦＩ		技術表示箇所
Ｇ０６Ｆ　5/00			Ｇ０６Ｆ　5/00	Ｈ	
Ｇ０６Ｔ　9/00		9382－5Ｋ	Ｈ０３Ｍ　7/46		
Ｈ０３Ｍ　7/46			Ｈ０４Ｎ　1/413	Ｚ	
Ｈ０４Ｎ　1/413			Ｇ０６Ｆ　15/66	３３０Ｅ	

審査請求　未請求　請求項の数２　ＯＬ　（全 10 頁）

(21)出願番号　特願平7－60900

(22)出願日　平成７年(1995)３月20日

(71)出願人　591044164
株式会社沖データ
東京都港区芝浦四丁目11番地22号

(72)発明者　▲高▼橋　芳典
東京都港区芝浦４丁目11番地22号　株式会社沖データ内

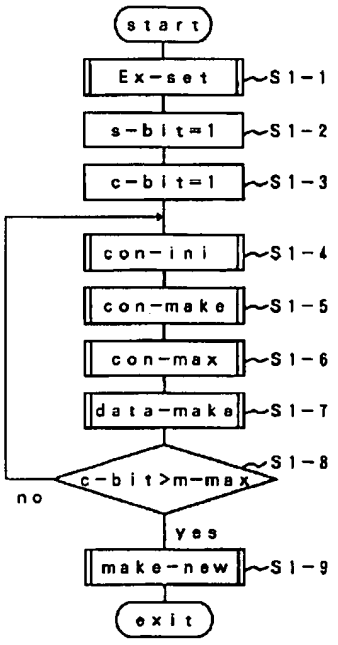
(74)代理人　弁理士　鈴木　敏明

(54)【発明の名称】　印刷データの圧縮方法

(57)【要約】

【目的】　変換を行う行のデータとその直前行のデータとがほとんど一致しない場合でも、効率的に圧縮が行える圧縮方法を提供する。

【構成】　まず変換を行う当該行とｎ行前の同じビット位置のデータの排他的論理和Ex(n,m)を求める（ステップ１－１）。そして未変換の先頭ビット位置変数s-bitおよび検索中ビット位置を示すポインタの変数c-bitの初期化を行って（ステップ１－２、１－３）、連続同様ビット数con(n)を初期化する（ステップ１－４）。次にサブルーチンcon-makeをコールして、未変換先頭ビット位置からの先行各行での連続同様ビット数を検索してその結果をcon(n)に格納する（ステップ１－５）。生成されたcon(n)のうち最大のものを探し（ステップ１－６）、結果として得られた最大連続同様ビット数にしたがって、サブルーチンdata-makeをコールして変換データを生成する（ステップ１－７）。



実施例の動作を示すフローチャート

1

【特許請求の範囲】

【請求項1】　印刷データを圧縮する方法において、

変換を行う当該行の印刷データと同様のデータが当該行に先行する複数の行にあるか否かを検索し、

前記先行する複数行の各々において連続する同様のデータの数を算出し、

前記先行する複数行のうち連続する同様のデータの数の最も多い最長連続同様データ保持行と前記当該行の連続する同様のデータ数が所定数以上同様である場合、前記保持行の同様データで前記当該行のデータを変換することを特徴とする印刷データの圧縮方法。

【請求項2】　前記最長連続同様データ保持行の同様データでの前記当該行のデータの変換は、該保持行の同様データで変換することを示すビットと、連続同様ビット数を示すビットと、該保持行の先行行数を示すビットとにより行う請求項1記載の印刷データの圧縮方法。

【発明の詳細な説明】

【0001】

【産業上の利用分野】本発明は、プリンタで印刷を行う印刷データを圧縮する方法に関し、とくに印刷データを保管する際に圧縮する方法に関する。

【0002】

【従来の技術】図2は印刷結果を示す説明図、図3は図2の印刷結果を得るための印刷データを示す説明図である。図2は印刷可能な最小物理サイズ素点（ドット）の集まりを表現したもので、図3はドットを2進値（1または0の情報、ビット）の集まりとして表現している。

【0003】図3において、左から12番目までの1、0の組み合わせは、第6行以降変化していない。そこで各々の行のデータを直前の行のデータと比較し、直前の行と同じビットの組み合わせの部分は、「ここからの連続するmビットの組み合わせは、直前行と同じである」という情報に変換し、直前の行とビットの組み合わせが異なる部分は、「ここからの連続するnビットは、続くnビットの情報を充てる」という情報として格納することにより、変換する前よりも少ないビット数で再生可能なビット列が生成できる。この様な圧縮方法では、続くデータが直前行と同じ情報を示すのか否か、あるいは何ビット分が直前行と同じなのかを示すためのビットが必要になる。

【0004】

【発明が解決しようとする課題】しかしながら上記従来の圧縮方法では、図4に示すようなドットに印刷する場合はデータを圧縮できない。図5は図4のドットに対応する2進値表である。図5からわかるように、各々の行のデータは直前行のデータとほとんど一致しない。したがって、上記従来の方法で圧縮すると、続くデータが直前行と同じ情報を示すのか否か、あるいは何ビット分が直前行と同じなのかを示すためのビットが必要になる分、ビット数が増加することになる。

2

【0005】

【課題を解決するための手段】上記課題を解決するために本発明の圧縮方法は、変換を行う当該行の印刷データと同様のデータが当該行に先行する複数の行にあるか否かを検索し、前記先行する複数行の各々において連続する同様のデータの数を算出し、前記先行する複数行のうち連続する同様のデータの数の最も多い最長連続同様データ保持行と前記当該行の連続する同様のデータ数が所定数以上同様である場合、前記保持行の同様データで前記当該行のデータを変換するようにしたものである。

【0006】

【作用】上記構成を有する本発明によれば、変換を行う当該行の印刷データと同様のデータが当該行に先行する複数の行にあるか否かを比較し、その中から、当該行の印刷データと同様の連続するデータを有する先行行のデータを指定して圧縮するので、効率的に圧縮を行うことができる。

【0007】

【実施例】以下、本発明に係る実施例を図面にしたがって説明する。なお各図面に共通する要素には同一の符号を付す。図1は本発明に係る実施例の動作を示すフローチャートである。

【0008】まず図5において、第3行のドット組み合わせは、直前の第2行と比較すると、第1列、2列、5～29の奇数列、30列および32列が同じであるが、第1行と比較すると、第4～第32列までのビット列が全く同じである。この様に、該当行に先行する複数行まで比較の範囲を広げることにより、「続くnビットはm行前の行の同じビット位置のデータと同じである」という情報に変換し、前の複数行まで比較してもなお該当行と同じビットの組み合わせが存在しない場合は、「ここからの連続するnビットは続くnビットの情報を充てる」という情報として格納することにより、変換する前よりも少ないビット数で再生可能なビット列が生成できる。

【0009】上記2種類の情報を混在させるために本実施例では図6、図7に示す様式を使用する。図6は様式①を示し、図7は様式②を示す。本実施例では、1つの様式で指定する最大ビット数は16、比較の対象とする先行行数は最大8としている。これら2つの数値は、変換の対象となる実際のドット列の組み合わせにより最も効率の良い値が異なる。

【0010】本実施例では、続くデータが先行行と同じ情報を示すのか否かを表すために1ビットを充てる。図6に示す様式①601において、“1”は先行行と同じ情報であることを示し、“0”は新たなビットの組み合わせであることを示す。例えば図6においては、様式識別ビット602は“1”となっているので、続くデータは先行行と同じ情報である。連続同様ビット数603は、格納される数値＋1で何ビット分が連続して先行行

と同じ組み合わせかを示す。また先行行数６０４は、格納される数値＋１で何行前と連続して同様なのかを示す。したがって、図に示すように、連続同様ビット数６０３が５で、先行行数６０４が３の場合は、６ビットデータが４行前の行と連続して同じ情報であることを示している。

【００１１】図７において、様式②７０１では、様式識別ビット７０２は“０”である。挿入ビット数７０３は、格納される数値＋１で何ビット分が連続して挿入されるかを示す。挿入データ７０４は、挿入されるべきデータそのものを示す。即ち図７に示すように、挿入ビット数７０３が４である場合、５ビットデータ７０４が連続して新たに挿入されることを示している。

【００１２】図５に示す２進値表の第１行目のデータは、比較する対象がないので、すべて様式②となり、したがって様式識別ビット７０２は“０”、挿入ビット数７０３は１５、挿入データ７０４は２進値で“0111010101010101”、続いて同じ様式②で挿入データ７０４は、“0101010101010111”となる。この結果、第１行のデータは図８に示す通りとなる。

【００１３】第２行目については、第１行と比較すると同様ビットが分散した４ビットのみなので、第１行と同じくすべて様式②となり、第２行データは図９に示す通りの結果となる。

【００１４】また第３行については、最初の２ビットは第２行と同様であるが、様式①で表現すると８ビットを要し、様式②の場合の７ビットよりもビット数が増加する。そこで第３ビット目を含めて様式②で表現する。第４ビット～第３２ビットはすべて第１行と同様なので様式①で表現する。第４ビット～第１９ビットは「連続する１６ビットは２行前と同様」という内容となるので、様式識別ビットは６０２は“１”、同様ビット数は１５、先行行数は１となる。第２０ビット～第３２ビットは「連続する１３ビットは２行前と同様」という内容となるので、様式識別ビットは６０２は“１”、同様ビット数は１２、先行行数は１となる。この結果、第３行データは図１０に示す通りとなる。

【００１５】ここで何ビット連続して同様の場合に様式②よりも様式①を採用した方が必要ビット数が少ないかを考えてみる。様式①は固定的に８ビットを必要とする。連続するｘビットを様式②で表現すると、ｘビットのデータそのものに５ビットが付加されるので、計（ｘ＋５）ビットが必要となる。このビット数が８以上、即ちｘが４以上、つまり４ビット以上が連続して先行行と同様の場合に様式①を採用すると、必要ビット数が様式②と同じかまたは様式②より少なくなる。

【００１６】図１１は本発明の実施例を適用するプリンタのブロック図である。同図において、ＣＰＵ１１１はプリンタに動作全体を制御するとともに、本実施例における印刷データの圧縮を行う。ＣＰＵ１１１にはバス１

１２を介してＲＯＭ１１３、ＲＡＭ１１４およびポート１１５が接続されている。ＲＯＭ１１３には制御プログラム等が格納され、ＲＡＭ１１４には印刷データが一時的に格納される。ポート１１５には、上位インタフェース回路１１６および印刷機構部駆動回路１１７が接続され、印刷機構部駆動回路１１７にはさらに印刷機構部１１８が接続されている。上位インタフェース回路１１６には図示しない上位装置が接続されており、この上位装置からプリンタに印刷データが送られてくる。

【００１７】次に各フローチャートにしたがって印刷データの圧縮動作を説明する。図１に示すフローチャートは、本実施例のアルゴリズム全体を示すメインルーチンであり、いくつかのステップにおいてサブルーチンを呼び出す。ここで、各サブルーチンで使用されるアルゴリズムについて説明する。

【００１８】まず、cur(m)は当該行の第mビットを表し、Bit(n,m)は当該行のｎ行前の第mビットを表す。つまりここでは、cur(m)とBit(n,m)には予め相当するビットデータが格納されているという前提にたっている。Ex(n,m) は当該行とｎ行前の同じビット位置のデータの排他的論理和（記号＃は排他的論理和の演算子である）であり、Ex(n,m) が０の場合、当該行の第mビットと同じデータがｎ行前にあるといえる。con(n)はｎ行前に検索中のビット位置から連続して存在する当該行と同じビットの組み合わせ数が格納される。s-bit は未変換の先頭ビット位置の変換であり、c-bit は検索中ビット位置を示すポインタの変数である。m-max は当該行の総ビット幅を表す。なお使用する様式は、図６および図７に示す様式①と様式②としているので、同様ビットを検索する先行行数６０４は８で、連続同様ビット数６０３および挿入ビット数７０３の最大値は１６としている。また、当該行に先行する行の数が８未満の場合の例外処理は以下に説明する動作には含まれていない。

【００１９】まず図１のフローチャートにおいて、サブルーチンEx-setを呼び出してEx(n,m) を生成する。図１２はサブルーチンEx-setを示すフローチャートである。図１２において、まず先行行のナンバーを表す変数ｉを１に初期化し（ステップ１２－１）、次にビット位置を表す変数ｊを１に初期化する（ステップ１２－２）。ステップ１２－３では、当該行と１行前の最初のビット位置のデータの排他的論理和Ex(i,j) を求め、ステップ１２－４でビット位置を変え、最後のビット位置のデータまで（ステップ１２－５）、当該行と１行前の各ビット位置のデータの排他的論理和Ex(i,j) を求める。当該行と１行前の各ビット位置のデータの排他的論理和Ex(i,j) が求められると、次に当該行と２行前の各ビット位置のデータの排他的論理和Ex(i,j) を求める（ステップ１２－６）。以上の処理を８行前の各ビット位置のデータについて行う（ステップ１２－７）。

【００２０】図１において、サブルーチンEx-setが終了

5

すると、未変換先頭ビット位置変数s-bit を初期化する（ステップ１－２）とともに、検索中ビット位置を示すポインタ変数c-bit の初期化を行う（ステップ１－３）。次にサブルーチンcon-iniを呼び出す（ステップ１－４）。

【００２１】図１３はサブルーチンcon-ini を示すフローチャートである。同図において、まず先行行のナンバーの変数ｉを１に初期化し（ステップ１３－１）、連続同様ビット数con(i)を初期化して０とする（ステップ１３－２）。以上の処理を先行する８行分行う（ステップ１３－３、１３－４）。これにより連続同様ビット数con(i)としてすべて０は格納される。この処理が終了すると、次にサブルーチンcon-makeをコールする（ステップ１－５）。

【００２２】図１４はサブルーチンcon-makeを示すフローチャートである。同図において、まず本サブルーチン内で使用するビット位置変数ｊを未変換先頭ビット位置変数s-bit で初期化する（ステップ１４－１）。これにより本サブルーチン内では、s-bit 位置より同様ビットの有無等が検索される。次に先行行のナンバーｉを１で初期化する（ステップ１４－２）。次にｉ番目の先行行の検索中ビット位置データが当該行の検索中ビットのデータと同じか否かを判断し（ステップ１４－３）、同じならばステップ１４－４に分岐し、同じでなければステップ１４－１１に分岐する。

【００２３】ステップ１４－４では、同様ビットが１つ見付かったので、検索中の連続同様ビット数con(i)を１とする。その後第ｉ先行行において何ビット連続して同様かを検索し、前記con(i)に格納する。ここでは連続同様ビット数検索中のビット位置を示す変数としてｋを用いている。即ち、検索中ビット位置は変数ｊに在るので、変数ｋはステップ１４－５でｊの値に初期化され、次にビット位置を調べるためにステップ１４－６で１増加させ、ステップ１４－９で実際に当該行と同じデータか否かを調べる。ステップ１４－９で同じと判断された場合は、ステップ１４－１０で連続同様ビット数con(i)を１増加させ、さらに次のビット位置の検索のためにステップ１４－６に無条件分岐する。ステップ１４－９で当該行と同じでないと判断された場合は、即座に検索ループを抜け、ステップ１４－１１に分岐して次の先行行での処理に移行する。

【００２４】ステップ１４－７では、検索中に連続同様ビット数がその最大値１６を越えた場合はループを抜けることを示す。また変数ｋはビット位置を表す変数なので、検索時のその値は最大ビット幅m-max を越える必要がなく、ステップ１４－８はビット位置変数ｋが最大ビット幅m-max を越えるとループを抜けることを示している。以上により、未変換先頭ビット位置からの先行各行での連続同様ビット数の検索結果がcon(n)に格納される。サブルーチンcon-makeの処理が終了すると、つぎにサブルーチンcon-max がコールされる（ステップ１－６）。

6

【００２５】図１５はサブルーチンcon-makeを示すフローチャートである。このサブルーチンは、各先行行の連続同様ビット数con(i)のうち最大のものを探す処理を行う。同図において、ｉは検索中の先行行のナンバーを示し、ｐは第ｉ先行行検索時点までの最大連続同様ビット数を有する行の先行行ナンバーを保持する変数である。まず先行行ナンバーの変数ｉを当該行の直前行を表す１に初期化し（ステップ１５－１）、最長連続同様ビット保持行ナンバーｐを０とする（ステップ１５－２）。これにより検索開始時点でいずれの行にも同様ビットが存在しないように設定する。次にステップ１５－３で、検索中の行の連続同様ビット数が０か否かを判断し、０の場合はステップ１５－７に分岐して次の先行行に検索の対象を移す。

【００２６】検索中の行の連続同様ビット数が０でない場合は、既知の最長連続同様ビット数との比較を行うべくステップ１５－４に分岐する。ステップ１５－４では、既知の最長連続同様ビット数が在るか否かを判断し（最長連続同様ビット保持行ナンバーｐが０の場合は、既知の最長連続同様ビット数は存在しない）、既知の最長連続同様ビット数が存在しない場合、ステップ１５－６に分岐して検索中の行を最長連続同様ビット保持行ナンバーとするために、ｐに検索中の行ナンバーｉを代入する。

【００２７】また既知の最長連続同様ビット数が存在する場合は、ステップ１５－５で、既知の最長連続同様ビット数con(p)と検索中の行の連続同様ビット数con(i)とを比較し、検索中の行の連続同様ビット数con(i)が既知の最長連続同様ビット数con(p)よりも大きい場合に、ステップ１５－６にて、検索中の行を最長連続同様ビット保持行ナンバーとするために、ｐに検索中の行のナンバーｉを代入する。検索中の行の連続同様ビット数con(i)が既知の最長連続同様ビット数con(p)よりも大きくない場合は、最長連続同様ビット保持行ナンバーｐは更新せず、ステップ１５－７に分岐して次の先行行に検索の対象を移す。ステップ１５－８は、検索対象の先行行の検索を終了したか否かを、検索中の行のナンバーｉが８を越えたか否かで判断し、検索対象行数以下の場合は次の先行行の検索のためにステップ１５－３に分岐し、検索対象行数８を越えた場合はこのサブルーチンを終了する。本サブルーチンの終了時、変数ｐには最長連続同様ビット保持行のナンバーが格納される。同様ビットをいずれの検索対象先行行にも有しない場合、変数ｐは０となっている。サブルーチンcon-max の処理が終了すると、つぎにサブルーチンdata-make がコールされる（ステップ１－７）。

【００２８】図１６はサブルーチンdata-make を示すフローチャートである。このサブルーチンdata-make は、

7

サブルーチンcon-max で得られた最長連続同様ビット数にしたがって、変換データを生成する処理を行う。このサブルーチンdata-make の説明に先立って、このサブルーチンdata-make の中でコールされる別のサブルーチンmake-same およびサブルーチンmake-newについて説明する。図１７はサブルーチンmake-same を示すフローチャートである。

【００２９】図１７において、サブルーチンmake-sameは、様式①にて先行行に連続同様データが在る場合の変換データを作成するサブルーチンである。なおこのサブルーチンにおいて、平行四辺形の処理ボックスは、変換データをメモリ（図１１に示すＲＡＭ１１４）内の領域に格納する処理を示す。まずステップ１７－１で、メモリ内に様式識別ビット６０２として“１”を格納する。これにより様式①であることを明示する。次にステップ１７－２で、検索中ビット位置を示すポインタ変数c-bit と未変換先頭ビット位置変数s-bit の差を連続同様ビット数６０３としてメモリに格納する。続くステップ１７－３では、様式①の先行行数６０４を格納するが、変数ｐは直前行のナンバーを１としているので、様式①に対応させるためにｐ－１を格納する。

【００３０】図１８はサブルーチンmake-newを示すフローチャートである。このサブルーチンmake-newは、いずれの検索対象先行行にも同様データが存在しない場合に、様式②で変換データを作成するサブルーチンである。このサブルーチンにおいても、平行四辺形の処理ボックスは、変換データをメモリ（図１１に示すＲＡＭ１１４）内の領域に格納する処理を示す。まずステップ１８－１で、メモリに様式識別ビット７０２として“０”を格納し、様式②であることを明示する。ステップ１８－２では、検索中ビット位置を示すポインタ変数c-bit と未変換先頭ビット位置変数s-bit の差を挿入ビット数７０３としてメモリに格納する。検索中ビット位置を示すポインタ変数c-bit を利用して、様式②の挿入ビット数７０３として (c-bit)-(s-bit)を充てることができる。

【００３１】ステップ１８－３からステップ１８－６では、様式②の挿入データ７０４を格納する。変数ｉは格納すべきビット位置のポインタとして使用されており、ステップ１８－３で未変換ビット位置変数s-bit を代入して初期化され、ステップ１８－４で当該行のビットデータcur(i)を格納し、ステップ１８－５で１ずつ増加されるポインタに従って、ステップ１８－６でポインタがc-bit を越えるまで連続してメモリに格納する。

【００３２】次に図１６のフローチャートにしたがってサブルーチンdata-make について説明する。このサブルーチンでは、連続同様ビット数con(p)が４を越える場合に、様式①で変換データを格納して未変換先頭ビット位置変数s-bit を更新し、それ以外の場合は、未変換ビット数 ((c-bit)-(s-bit+1)) が様式②で規定される最大

8

ビット長１６を越えようとする場合に様式②で変換データを格納して未変換先頭ビット位置変数s-bit を更新し、越えない場合は単にc-bit を１増加している。

【００３３】まずステップ１６－１では、検索対象先行行に同様ビットが存在するか否かを、変数ｐが０（存在しない場合）か１（存在する場合）かで判定し、存在する場合はステップ１６－２に分岐し、存在しない場合はステップ１６－３に分岐する。ステップ１６－２では、前述したように連続同様ビット数が４以下の場合は様式②の方が変換後のビット数が少なくなるので、連続同様ビット数が４以下の場合はステップ１６－３に分岐して同様ビットが存在しない場合と同様の処理を行う。

【００３４】ステップ１６－２で連続同様ビット数が４を越える場合は、ステップ１６－４に分岐し、未変換先頭ビット位置と検索中ビット位置が等しいか否かを判断する。ここで等しくないと判断された場合は、未変換先頭ビット位置と検索中ビットの１ビット前まではデータは様式②に従って変換されなければならないので、ステップ１６－５からステップ１６－８にて様式②に従って変換される。ステップ１６－５では、検索中ビット位置の１ビット前までのデータを様式②に従って変換するために、検索中ビット位置ポインタ変数を１減らす。これにより次にステップ１６－６でサブルーチンmake-newを呼び出す準備をする。ステップ１６－６ではサブルーチンmake-newを呼び出し、図１８で説明したサブルーチンmake-new処理を行い、様式②に従ってデータを変換する。ステップ１６－７では、ステップ１６－５で補正したc-bit の内容を復旧し、ステップ１６－８で、復旧したc-bit の値をs-bit に代入して未変換先頭ビット位置を補正する。

【００３５】ステップ１６－４で、未変換先頭ビット位置と検索中ビット位置が等しいと判断された場合は、ステップ１６－９に分岐する。ステップ１６－９では、次のステップ１６－１０で呼び出されるサブルーチンmake-same 処理で使用される変数c-bit の意味に合わせるために、c-bit にcon(p)を加算してさらに１減ずることにより、c-bit の内容を同様ビット後縁位置としている。ステップ１６－１０では、サブルーチンmake-same を呼び出して、図１７で説明したように、様式①の変換データを格納し、ステップ１６－１１およびステップ１６－１２で、s-bitおよびc-bit を更新する。

【００３６】ステップ１６－３では、検索中ビット位置を示すポインタ変数c-bit が未変換先頭ビット位置変数s-bit に対して１５多い（１６ビット連続して連続同様ビットが存在しない）場合には、その時点の検索中行ビットまでのデータを変換するために、ステップ１６－１３に分岐し、サブルーチンmake-newを呼び出して、様式②に従ってデータを変換する。そしてステップ１６－１４およびステップ１６－１６で、s-bit およびc-bit を更新する。

9

【００３７】ステップ１６－３でc-bit とs-bit の差が１５未満の場合は、ステップ１６－１５に分岐し、検索中ビット位置を示すポインタ変数c-bit が最大ビット幅に達しているか否かを判断し、達している場合は処理を終了し、達していない場合はステップ１６－１６に分岐してc-bit を更新する。

【００３８】図１におけるステップ１－８は、未変換ビットが最大ビット幅に達した場合にループを抜けるためのステップであり、未変換ビットが最大ビット幅に達した場合ステップ１－９に分岐し、それ以外の場合はステップ１－４に分岐する。ステップ１－９では、未変換ビットがある場合にそれらを様式②に変換するサブルーチンmake-newを呼び出し、処理を行う。

【００３９】以上述べた変換アルゴリズムに基いて図５に示す２進値表を変換した結果を図１９に示す。本実施例により変換した場合は、９６０ビットのデータが６９６ビットに変換され、約７２％に圧縮される。

【００４０】

【発明の効果】以上詳細に説明したように本発明によれば、当該行のデータと先行する複数行のデータとを比較し、その中から当該行と同様のビットを最も長く有する行データを指定して圧縮するので、直前行のデータとほとんど一致しない印刷データもビット数が少なくなるように効率的に圧縮することができる。

【図面の簡単な説明】

【図１】本発明の実施例の動作を示すフローチャートである。

【図２】印刷結果を示す説明図である。

【図３】図２の印刷結果を得るための印刷データを示す説明図である。

【図４】他の印刷結果を示す説明図である。

10

【図５】図４の印刷データを示す説明図である。

【図６】圧縮データの様式①を示す説明図である。

【図７】圧縮データの様式②を示す説明図である。

【図８】第１行データを示す説明図である。

【図９】第２行データを示す説明図である。

【図１０】第３行データを示す説明図である。

【図１１】実施例を適用するプリンタのブロック図である。

【図１２】実施例のサブルーチンを示すフローチャートである。

【図１３】実施例のサブルーチンを示すフローチャートである。

【図１４】実施例のサブルーチンを示すフローチャートである。

【図１５】実施例のサブルーチンを示すフローチャートである。

【図１６】実施例のサブルーチンを示すフローチャートである。

【図１７】実施例のサブルーチンを示すフローチャートである。

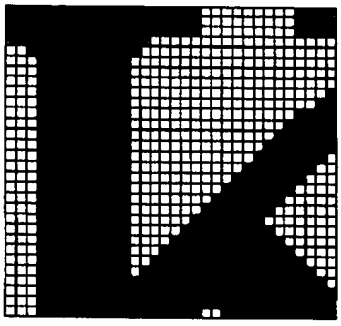
【図１８】実施例のサブルーチンを示すフローチャートである。

【図１９】実施例により変換した結果を示す説明図である。

【符号の説明】

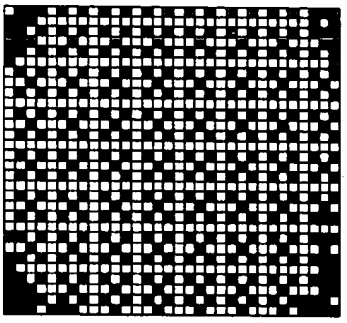
１１１　ＣＰＵ
１１４　ＲＡＭ
６０１　様式①
６０２　様式識別ビット
６０３　連続同様ビット数
６０４　先行行数

【図２】



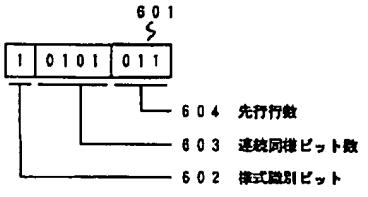
印刷結果を示す説明図

【図４】



他の印刷結果を示す説明図

【図６】

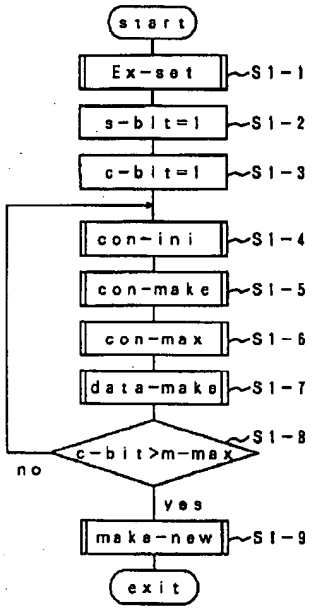


圧縮データの様式①を示す説明図

【図１０】

第３行データを示す説明図

【図1】



実施例の動作を示すフローチャート

【図3】

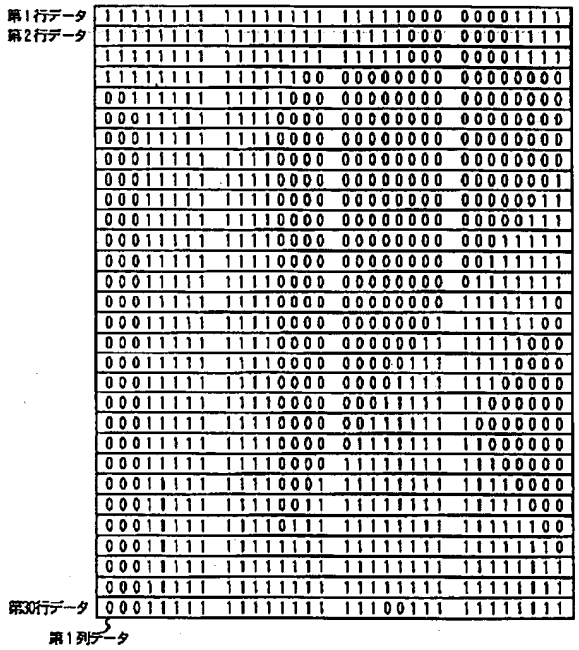


図2の印刷結果を得るための印刷データを示す説明図

【図5】

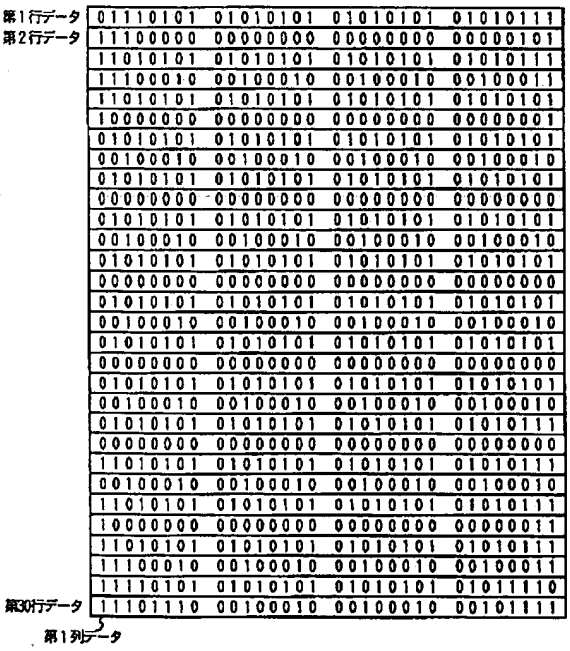
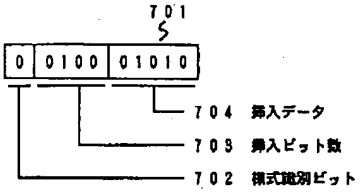


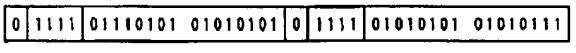
図4の印刷データを示す説明図

【図7】



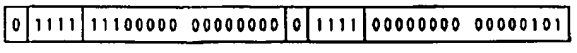
圧縮データの様式②を示す説明図

【図8】



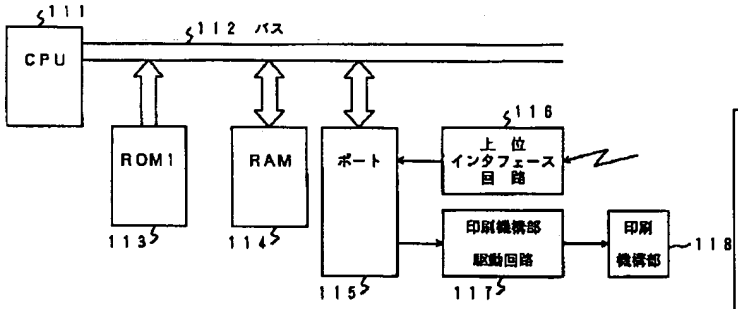
第1行データを示す説明図

【図9】



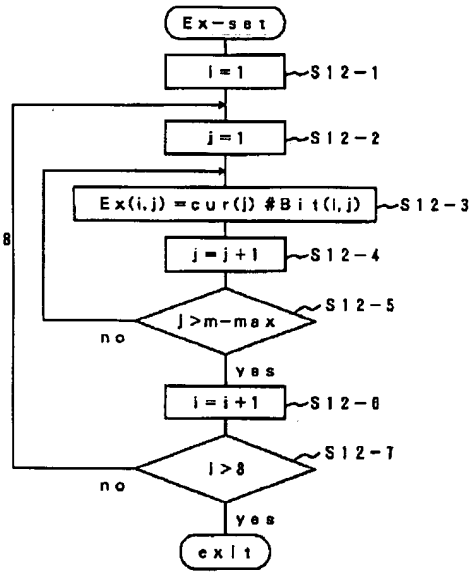
第2行データを示す説明図

【図１１】



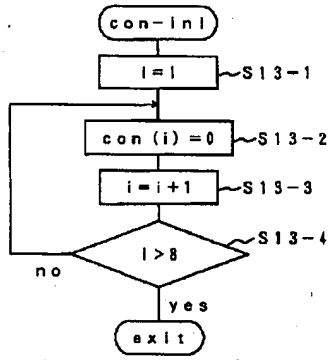
実施例を適用するプリンタのブロック図

【図１２】



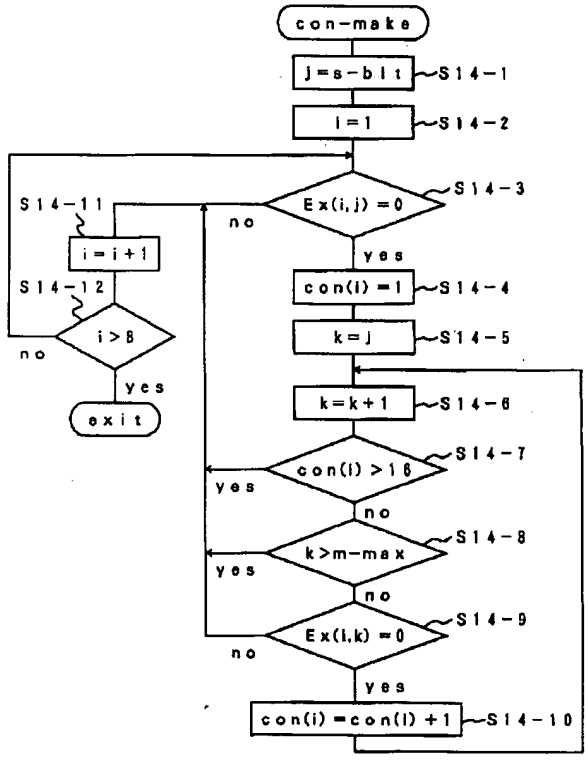
実施例のサブルーチンを示すフローチャート

【図１３】



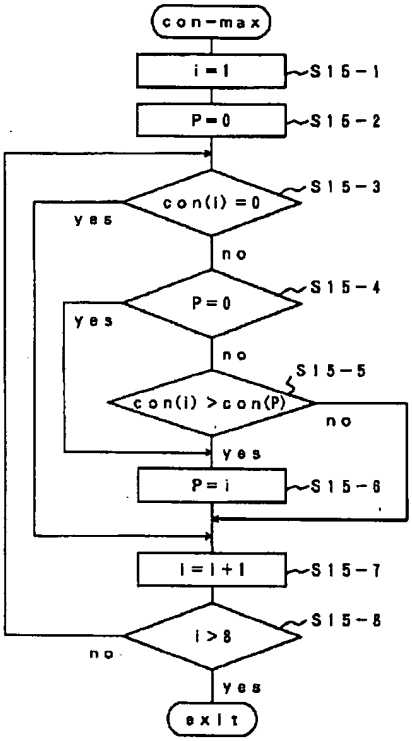
実施例のサブルーチンを示すフローチャート

【図１４】



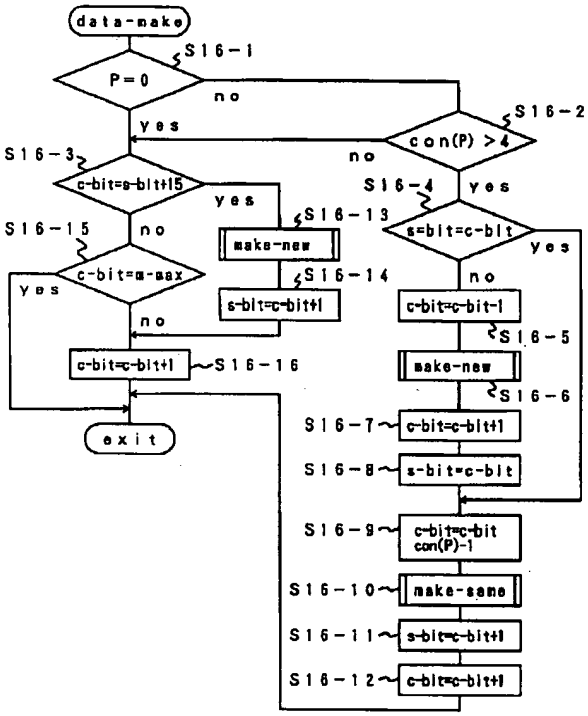
実施例のサブルーチンを示すフローチャート

【図15】



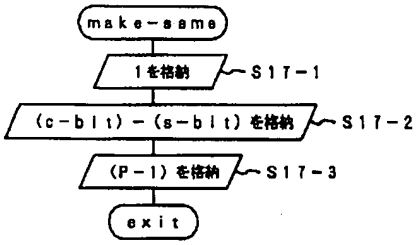
実施例のサブルーチンを示すフローチャート

【図16】



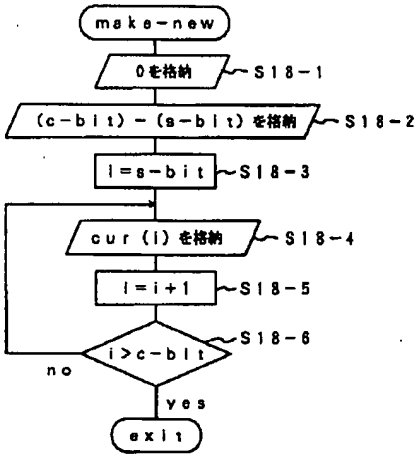
実施例のサブルーチンを示すフローチャート

【図17】



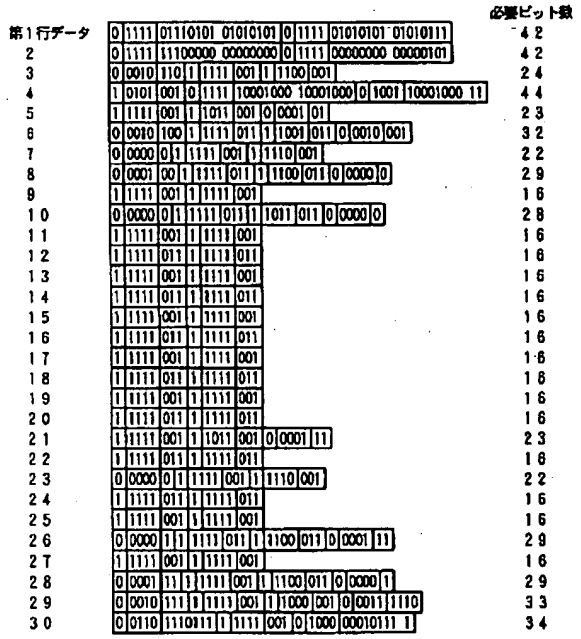
実施例のサブルーチンを示すフローチャート

【図18】



実施例のサブルーチンを示すフローチャート

【図１９】



実施例により変換した結果を示す説明図